CS 1322 Test 5 - Version 5 Thursday April 10^{th}

Name: _____

GT Number: _____

TA: _____

Commenting your code on this test is optional, however it may help the grader understand your code better. Other style issues such as using good variables names and using abstraction are still required. Write your gt number in the spaces provided and color in the corresponding numbers and letters.

gt	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	d	e	f	g	h	0	1	2	3	4	5	6	7	8	9	-	-	-	-	-	-	-	-	-	-
	0	1	2	3	4	5	6	7	8	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	0	1	2	3	4	5	6	7	8	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	0	1	2	3	4	5	6	7	8	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Problem	Points Possible	Points Received	Graded By
1 Matching	5		
2 Tracing	15		
3 Exceptions	15		
4 File I/O	15		
5 Stacks and Queues	10		
6 Heaps	15		
7 Linked Lists	10		
8 Binary Search Trees	15		
TOTAL	100		

Please remember: Any academic misconduct (including, but not limited to, the list below) could result in a 0 (zero) on the exam and/or an F grade in the course:

- Communication with anyone other than a proctor for ANY reason in ANY language in any manner.
- Sharing of ANYTHING (e.g. pencils, paper, erasers).
- Writing on paper that is not given to you by a proctor.
- Failure to follow directions given by the proctor.
- Failure to stop writing when the allowed time is up (as reported by the proctor).
- Use of cell phones, beepers, handheld computers, calculators, during the exam.
- Using books or other reference material.
- Disruption of the exam setting.

By taking this exam, you signify that it is your work and that you have neither given nor received inappropriate help during the taking of this exam in compliance with the Academic Honor Code of Georgia Tech.

Signature: _____

(you must sign this for your exam to be graded!)

1. Matching [5 pts]

Choose the **best** definition for each of the words below.

1. _____ A LIFO data structure that can be implemented with a Linked List.
2. _____ A data structure where every parent node is greater than or equal to the values of either of its child nodes.
3. _____ An exception that the compiler does not require the programmer to keep track of it.
4. _____ An abstraction for a sequence of bytes which data can be read or to which data can be written.
5. _____ Provides layers of functionality to I/O classes in Java.

A. Stack
B. Full Binary Tree
C. Adornation Class
D. Min-Heap
E. Queue
F. Unchecked Exception
G. Checked Exception
H. Error Exception
I. Max-Heap
J. Priority Node
K. Stream
L. Decorator Class
M. Complete Binary Tree

2. Tracing [15 pts]

What is the output of following code when the main method is run?

```
public class TracerB{

  public static void main(String[] args){
    int x=0;
    int y=2;
    double temp=2.0d;
    try{
      System.out.println("I'm going in.");
      while(y>=0){
          System.out.println("In the while loop");
          System.out.print(y/x);
          temp+=1.0d;
      }
      System.out.println("I'm out");
    }
    catch(ArithmeticException ae){
      System.out.println("I can't do that.");
      temp=4.0d;
    }//end catch(ArithmeticException)
    catch(Exception e){
      System.out.println(temp);
      temp=3.1415;
    }//end catch(Exception)
    finally{
      System.out.println("Final value: "+temp);
    }//end finally
    System.out.println("All Done");
  }//end main(String[])
}//end TracerB class
```

3. Exceptions [15 pts]

5 (a) Write a custom checked Exception called **TooRandomException**. You do not have to write any methods for this class.

10 (b) Write a method called **public void evaluateSize(int a)** inside the **Calculator** class that takes in an int and throws a **TooRandomException** if the number passed in is equal to 53, or otherwise prints the number. Remember to write the method header.

```
public class Calculator{




}
```

4. **File I/O [15 pts]**

Given the following class MyIO write the method **public void readAndPrint**, which reads all the lines from System.in until the end of file (this works the same as reading from any file) and prints them out to System.out:

```
public class MyIO{

public void readAndPrint(){









    }
} //end class MyIO
```

Hint: You might want to use the class BufferedReader and the method readLine() in the BufferedReader class.

5. Stacks [10 pts]

Assume you have the following **LinkedList** class. You may only assume that you have the following methods in the LinkedList class:

```
public class LinkedList{
  private LLNode head; // Beginning reference of LL
  public void addToFront(Object o){/*Code omitted for brevity */ }
  public void addToBack(Object o){/*Code omitted for brevity */ }
  public Object removeFromFront(){/*Code omitted for brevity */ }
  public Object removeFromBack(){/*Code omitted for brevity */ }
}
```

Write the following methods for the **Stack** class. The methods should maintain the intergrity of the Stack data structure.

```
public class Stack extends LinkedList {
```

[5] (a) **public void push(Object o){**

}

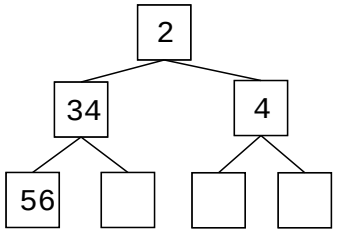
[5] (b) **public Object pop(){**

}

}//end class Stack

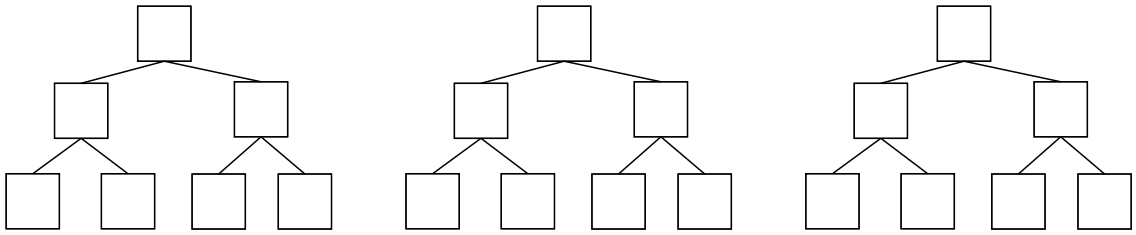
6. Heaps [15 pts]

Given the following Heap data structure:

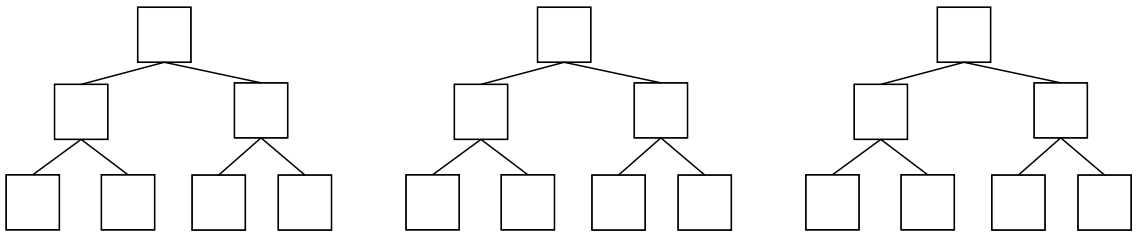


Show the tree data structure at each step of the process when:

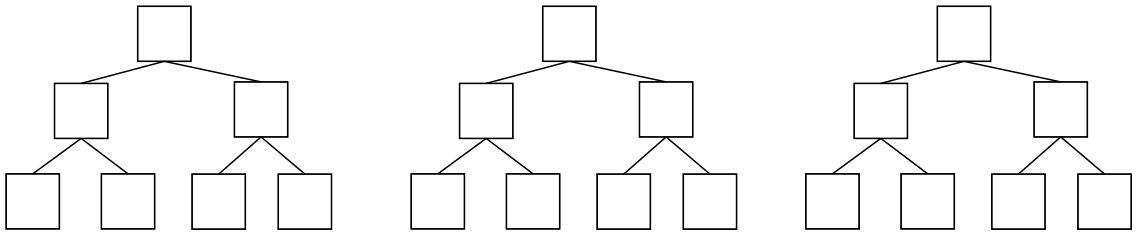
5 (a) The number 1 is added to the **original Heap**.



5 (b) The number 7 is added to the **original Heap**.



5 (c) The number 2 is removed from the **original Heap**.



7. LinkedList [10 pts]

Assume you have a fully functioning Linked List Node class called LLNode. With the following methods and constructor:

- `public void setData(Object o)`
- `public Object getData()`
- `public void setNext(LLNode next)`
- `public LLNode getNext()`
- `public LLNode(Object o) //set the data of the LLNode to o`

Write a method called **public void removeEveryOtherOne(Object o)** which remove every other occurrence of the Object o in the LinkedList. You may use any helper methods you wish to write.

```
public class LinkedList{
        private LLNode head;


        public void removeEveryOtherOne(Object o){









        }
} //end LinkedList class
```

8. Binary Search Trees [12 pts]

Given a fully functioning Binary Search Tree node class called **BSTNode** with the following constructor and methods:

- `public BSTNode(Comparable c)  //sets the data of the node to c`
- `public void setData(Comparable c)`
- `public Comparable getData()`
- `public void setRight(BSTNode rightChild)`
- `public BSTNode getRight()`
- `public void setLeft(BSTNode leftChild)`
- `public BSTNode getLeft()`

Write the method **Comparable findGreatestButNotGreaterThan(Comparable c)**which takes in a Comparable, and finds and returns a Comparable in the BST which the largest value inside the tree that is not greater than the Comparable c. If there is no data that satisfies the criteria, return null.. There should be no duplicated data inside the tree. You can only assume that you have the listed methods in the BSTNode to use. You may write any helper methods that you see fit.

```
public class BST {
       private BSTNode root;
```

public Comparable findGreatestButNotGreaterThan(Comparable c) {

/* More space on the next page */

```
/* Extra space for previous questions */
```

```
        } //end Comparable findGreatestButNotGreaterThan(Comparable c)
}//end BST class
```